Package: investlabr (via r-universe)

October 15, 2026

Title Research Workflows for Macro and Cross-Asset Analysis

Version 0.0.6.2

Author Oliver Zhou [aut, cre]

Maintainer Oliver Zhou <oliver.yxzhou@gmail.com>

Description Reusable research workflows for macro-financial and cross-asset analysis. The package sits on top of data packages such as 'investdatar' and focuses on transforming existing market and macro data into analysis-ready research tables, factor and indicator constructions, event and regime overlays, sensitivity estimates, scenario comparisons, and research-grade visualization outputs. Portfolio risk-return comparisons use the mean-variance framework of Markowitz (1952) <doi:10.1111/j.1540-6261.1952.tb01525.x>. Research-side volatility simulations include the generalized autoregressive conditional heteroskedasticity model of Bollerslev (1986) <doi:10.1016/0304-4076(86)90063-1>.

License MIT + file LICENSE

URL https://github.com/OliverLDS/investlabr

BugReports https://github.com/OliverLDS/investlabr/issues

Encoding UTF-8

Depends R (>= 4.1.0)

Imports data.table (>= 1.12.4), ggplot2 (>= 3.5.0), ggrepel, grid, gridExtra, jsonlite, patchwork, quadprog, rlang, scales, stats, utils, yaml, Rcpp

LinkingTo Rcpp

Suggests svglite, testthat (>= 3.0.0)

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Repository https://cran.r-universe.dev

Date/Publication 2026-10-15 19:17:18 UTC

RemoteUrl https://github.com/cran/investlabr

RemoteRef HEAD

RemoteSha 0424180883e7fa6a572c003e35dee7fe414c21d6

Contents

brief_data_as_of . . . 3
brief_plot_registry_entry . . . 4
brief_plot_registry_entry_v3 . . . 5
brief_plot_registry_validate . . . 8
brief_plot_registry_write . . . 9
brief_plot_registry_write_from_meta . . . 10
brief_score_label . . . 12
eval_event_performance . . . 12
eval_portfolio_performance . . . 13
eval_strat_performance . . . 14
eval_strat_plot_scatter_maxdd_annret . . . 16
eval_strat_plot_tsline_eq . . . 17
factor_bounded_score . . . 18
gen_candle_plots_with_sr_dts . . . 19
gen_candle_plots_with_sr_lines . . . 20
gen_facet_plot_from_multicol_ts . . . 22
gen_grid_of_plots_with_labels . . . 23
gen_plot_comparing_events . . . 24
gen_plot_event_tsline_cum_ret . . . 25
gen_rolling_monthly_stats . . . 26
gen_yield_curve_plot . . . 27
gen_yield_curve_plot_grid . . . 28
get_optimal_weights . . . 29
get_yield_data_DT . . . 29
prep_backtest_result_from_equity . . . 30
prep_fill_forward . . . 31
prep_ratio_ohlc . . . 32
prep_recent_changes . . . 33
prep_series_wide . . . 33
prep_zscore . . . 34
sim_ar1_dgp . . . 35
sim_ar2_dgp . . . 36
sim_ar3_dgp . . . 37
sim_arch1_dgp . . . 38
sim_forward_fan . . . 39
sim_garch11_dgp . . . 40
sim_ma1_dgp . . . 41
sim_scenario_path . . . 41
viz_annotate_event_lines . . . 42
viz_annotate_last_value . . . 44

viz_annotate_regime_bands 45
viz_annotate_source_caption 46
viz_candle_base 47
viz_forward_fan 48
viz_palette_get 49
viz_series_lines 50
viz_style_get 51
viz_style_set 52
viz_theme_apply 53

Index **55**

`brief_data_as_of` *Calculate a conservative artifact data date*

Description

Calculates the minimum of the latest usable observation date for every required input series. This prevents a high-frequency series from making a mixed-frequency artifact appear fresher than its stalest required input.

Usage

```
brief_data_as_of(series, date_col = "date", completed_before = NULL)
```

Arguments

`series` Named list of date vectors or data frames containing a date column. Inputs should already exclude observations that were not actually consumed by the artifact.

`date_col` Date-column name used for data-frame inputs.

`completed_before` Optional exclusive upper date bound. For Yahoo data, pass the current UTC date to exclude a potentially incomplete current-day bar.

Value

An ISO date string representing the latest common usable observation.

Examples

```
data_date <- brief_data_as_of(list(
  daily = as.Date(c("2025-01-06", "2025-01-07")),
  weekly = as.Date(c("2025-01-01", "2025-01-08"))
))
entry <- brief_plot_registry_entry_v3(
  id = "example-board", title = "Illustrative research board", collection = "macro",
  asset_class = "Rates", indicator_family = "Yield Curve", region = "United States",
  frequency = "Daily", source = "Synthetic data", expected_cadence = "weekly",
```

```
  rendered_at = "2025-01-09T12:00:00Z", data_as_of = data_date,
  metadata_updated_at = "2025-01-01", status = "ready",
 plot_image = "plots/example.svg", thumbnail = "thumbnails/example.png", tags = "example"
)
# This example validates metadata without requiring image files.
path <- tempfile(fileext = ".json")
brief_plot_registry_write(list(entry), path, schema_version = "3.0",
                          generated_at = "2025-01-09T12:01:00Z")
brief_plot_registry_validate(path, require_assets = FALSE)
unlink(path)
```

`brief_plot_registry_entry`

Build a deprecated schema 2.0 research-artifact registry entry

---

Description

This constructor is retained for source compatibility. New code should use `brief_plot_registry_entry_v3()`. Legacy dates are never interpreted as artifact render times or editorial metadata dates.

Usage

```
brief_plot_registry_entry(
  id,
  title,
  collection = NULL,
  asset_class,
  indicator_family,
  region,
  frequency,
  source,
  last_updated,
  status = NULL,
  plot_image,
  thumbnail,
  tags,
  subtitle = "",
  summary = "",
  description_md = "",
  section = "",
  source_detail = character(),
  curation_priority = 0,
  plot_html = "",
  related_ids = character(),
  compliance = list(educational_only = TRUE, not_investment_advice = TRUE),
  dashboard = NULL,
  published = NULL,
  featured = NULL,
```

```
  sort_priority = NULL,
  canonical_slug = NULL,
  podcast_topics = NULL,
  report_topics = NULL
)
```

Arguments

`id`, `title`, `asset_class`, `indicator_family`, `region`, `frequency`, `source`, `plot_image`, `thumbnail`, `tags`
: Core registry fields.

`collection`, `status`
: Nullable schema 2.0 fields. The deprecated `dashboard` and `published` arguments are used as fallbacks.

`last_updated`
: Legacy schema 2.0 artifact date.

`subtitle`, `summary`, `description_md`, `section`, `source_detail`, `curation_priority`, `plot_html`, `related_ids`, `compliance`
: Optional schema 2.0 fields.

`dashboard`, `published`, `featured`, `sort_priority`, `canonical_slug`, `podcast_topics`, `report_topics`
: Deprecated schema 1.0 compatibility fields.

Value

A validated schema 2.0 entry that cannot be emitted as schema 3.0.

Examples

```
# For existing schema 2 callers only; the constructor warns about deprecation.
entry <- suppressWarnings(brief_plot_registry_entry(
  id = "legacy-board", title = "Legacy example", collection = "macro",
  asset_class = "Rates", indicator_family = "Yield Curve", region = "United States",
  frequency = "Daily", source = "Synthetic data", last_updated = "2025-01-07",
  status = "ready", plot_image = "plots/example.svg",
  thumbnail = "thumbnails/example.png", tags = "example"
))
entry$last_updated
```

---

brief_plot_registry_entry_v3

Build one canonical schema 3.0 research-artifact registry entry

---

Description

Build one canonical schema 3.0 research-artifact registry entry

Usage

```
brief_plot_registry_entry_v3(
  id,
  title,
  collection,
  asset_class,
  indicator_family,
  region,
  frequency,
  source,
  rendered_at = NULL,
  data_as_of = NULL,
  metadata_updated_at,
  time_indexed = TRUE,
  expected_cadence,
  status,
  plot_image,
  thumbnail,
  tags,
  subtitle = "",
  summary = "",
  description_md = "",
  section = "",
  source_detail = character(),
  curation_priority = 0,
  plot_html = "",
  related_ids = character(),
  compliance = list(educational_only = TRUE, not_investment_advice = TRUE)
)
```

Arguments

`id`	Stable machine-safe plot identifier in lowercase kebab-case.
`title`	Reader-facing plot title.
`collection`	High-level research collection. One of `"macro"`, `"markets"`, `"crypto"`, or `"strategies"`.
`asset_class`	Asset-class label such as `"Rates"`.
`indicator_family`	Indicator-family label.
`region`	Region label.
`frequency`	Frequency label.
`source`	Source label.
`rendered_at`	UTC artifact-render timestamp in `YYYY-MM-DDTHH:MM:SSZ` format. Required for ready artifacts.
`data_as_of`	Latest observation actually represented by the artifact, as an ISO date. Required for ready time-indexed artifacts.

`metadata_updated_at`
: ISO date of the latest material human-authored metadata change.

`time_indexed`
: Whether the artifact contains time-indexed data.

`expected_cadence`
: Expected update cadence of `data_as_of`. One of `"daily"`, `"weekly"`, `"monthly"`, `"event_driven"`, or `"not_time_indexed"`.

`status`
: Artifact readiness: `"draft"`, `"ready"`, or `"archived"`.

`plot_image`
: Relative primary-plot path.

`thumbnail`
: Relative thumbnail path.

`tags`
: Flat character vector of tags.

`subtitle`, `summary`, `description_md`, `section`
: Optional descriptive fields.

`source_detail`
: Optional source details.

`curation_priority`
: Integer curation priority.

`plot_html`
: Optional relative HTML path.

`related_ids`
: Related artifact ids.

`compliance`
: Named compliance flags.

Value

A canonical schema 3.0 entry. Writers may project it to schema 2.0.

Examples

```
data_date <- brief_data_as_of(list(
  daily = as.Date(c("2025-01-06", "2025-01-07")),
  weekly = as.Date(c("2025-01-01", "2025-01-08"))
))
entry <- brief_plot_registry_entry_v3(
  id = "example-board", title = "Illustrative research board", collection = "macro",
  asset_class = "Rates", indicator_family = "Yield Curve", region = "United States",
  frequency = "Daily", source = "Synthetic data", expected_cadence = "weekly",
  rendered_at = "2025-01-09T12:00:00Z", data_as_of = data_date,
  metadata_updated_at = "2025-01-01", status = "ready",
  plot_image = "plots/example.svg", thumbnail = "thumbnails/example.png", tags = "example"
)
# This example validates metadata without requiring image files.
path <- tempfile(fileext = ".json")
brief_plot_registry_write(list(entry), path, schema_version = "3.0",
                          generated_at = "2025-01-09T12:01:00Z")
brief_plot_registry_validate(path, require_assets = FALSE)
unlink(path)
```

brief_plot_registry_validate

Validate a research-artifact plot registry

Description

Validate a research-artifact plot registry

Usage

```
brief_plot_registry_validate(
  registry,
  output_root = NULL,
  require_assets = TRUE
)
```

Arguments

registry	Registry object or JSON path. Schemas 1.0, 2.0, and 3.0 are readable; schema 2.0 remains the default writer output during migration.
output_root	Root for relative asset validation.
require_assets	Require referenced assets to exist.

Value

Named validation summary.

Examples

```
data_date <- brief_data_as_of(list(
  daily = as.Date(c("2025-01-06", "2025-01-07")),
  weekly = as.Date(c("2025-01-01", "2025-01-08"))
))
entry <- brief_plot_registry_entry_v3(
  id = "example-board", title = "Illustrative research board", collection = "macro",
  asset_class = "Rates", indicator_family = "Yield Curve", region = "United States",
  frequency = "Daily", source = "Synthetic data", expected_cadence = "weekly",
  rendered_at = "2025-01-09T12:00:00Z", data_as_of = data_date,
  metadata_updated_at = "2025-01-01", status = "ready",
 plot_image = "plots/example.svg", thumbnail = "thumbnails/example.png", tags = "example"
)
# This example validates metadata without requiring image files.
path <- tempfile(fileext = ".json")
brief_plot_registry_write(list(entry), path, schema_version = "3.0",
                          generated_at = "2025-01-09T12:01:00Z")
brief_plot_registry_validate(path, require_assets = FALSE)
unlink(path)
```

brief_plot_registry_write

Write a research-artifact plot registry to JSON

Description

Write a research-artifact plot registry to JSON

Usage

```
brief_plot_registry_write(
  plots,
  path,
  schema_version = "2.0",
  generated_at = Sys.time(),
  source_system = "investlabr",
  base_path_mode = "relative",
  ready_only = FALSE,
  pretty = TRUE,
  published_only = NULL
)
```

Arguments

`plots`	Schema 2.0 entries from `brief_plot_registry_entry()` or canonical schema 3.0 entries from `brief_plot_registry_entry_v3()`.
`path`	Explicit output JSON path. No file is written unless a path is supplied; examples should use a location under `tempdir()`.
`schema_version`	Output schema. Defaults to compatibility schema 2.0; use `"3.0"` explicitly for migration testing.
`generated_at`	Registry-generation timestamp in UTC.
`source_system`	Source-system label.
`base_path_mode`	Must be `"relative"`.
`ready_only`	Keep only ready entries.
`pretty`	Write pretty JSON.
`published_only`	Deprecated alias for `ready_only`.

Value

Invisibly returns the registry written to disk.

Examples

```
data_date <- brief_data_as_of(list(
  daily = as.Date(c("2025-01-06", "2025-01-07")),
  weekly = as.Date(c("2025-01-01", "2025-01-08"))
))
entry <- brief_plot_registry_entry_v3(
  id = "example-board", title = "Illustrative research board", collection = "macro",
  asset_class = "Rates", indicator_family = "Yield Curve", region = "United States",
  frequency = "Daily", source = "Synthetic data", expected_cadence = "weekly",
  rendered_at = "2025-01-09T12:00:00Z", data_as_of = data_date,
  metadata_updated_at = "2025-01-01", status = "ready",
 plot_image = "plots/example.svg", thumbnail = "thumbnails/example.png", tags = "example"
)
# This example validates metadata without requiring image files.
path <- tempfile(fileext = ".json")
brief_plot_registry_write(list(entry), path, schema_version = "3.0",
                          generated_at = "2025-01-09T12:01:00Z")
brief_plot_registry_validate(path, require_assets = FALSE)
unlink(path)
```

---

brief_plot_registry_write_from_meta

Build and write a research-artifact registry from metadata sidecars

---

Description

Build and write a research-artifact registry from metadata sidecars

Usage

```
brief_plot_registry_write_from_meta(
  meta_dir = file.path("config", "publishing", "plots"),
  path,
  output_root = dirname(path),
  resolved_meta_dir = file.path(output_root, "resolved"),
  schema_version = "2.0",
  source_system = "investlabr",
  base_path_mode = "relative",
  ready_only = FALSE,
  pretty = TRUE,
  published_only = NULL
)
```

Arguments

`meta_dir`	Tracked YAML sidecar directory.
`path`	Explicit output JSON path. No file is written unless a path is supplied; examples should use a location under `tempdir()`.

output_root	Root for relative asset paths.
resolved_meta_dir	Run-local metadata emitted by the renderer.
schema_version	Output schema. Defaults to compatibility schema 2.0; use "3.0" explicitly for migration testing.
source_system	Source-system label.
base_path_mode	Must be "relative".
ready_only	Keep only ready entries.
pretty	Write pretty JSON.
published_only	Deprecated alias for ready_only.

Value

Invisibly returns the registry written to disk.

Examples

```
local({
  root <- tempfile("registry-example-")
  dir.create(root)
  on.exit(unlink(root, recursive = TRUE))
  dir.create(file.path(root, "meta"))
  dir.create(file.path(root, "resolved"))
  # Use actual temporary plot artifacts, with explicit synthetic metadata.
  pdf_path <- file.path(root, "example.pdf")
  grDevices::pdf(pdf_path)
  plot(1:3)
  grDevices::dev.off()
  tracked <- list(
    id = "example-board", title = "Example", collection = "macro",
    asset_class = "Rates", indicator_family = "Example", region = "Global",
    frequency = "Event", source = "Synthetic data", status = "ready",
    metadata_updated_at = "2025-01-01", time_indexed = FALSE,
    expected_cadence = "not_time_indexed", plot_image = "example.pdf",
    thumbnail = "example.pdf", tags = "example"
  )
  yaml::write_yaml(tracked, file.path(root, "meta", "example.yaml"))
  jsonlite::write_json(list(
    id = tracked$id, rendered_at = format(Sys.time(), "%Y-%m-%dT%H:%M:%SZ", tz = "UTC"),
    data_as_of = NULL, metadata_updated_at = tracked$metadata_updated_at,
    time_indexed = FALSE, expected_cadence = "not_time_indexed",
    data_as_of_rule = "not_time_indexed"
  ), file.path(root, "resolved", "example-board.json"), auto_unbox = TRUE, null = "null")
  brief_plot_registry_write_from_meta(
    file.path(root, "meta"), file.path(root, "registry.json"), output_root = root,
    schema_version = "3.0"
  )
})
```

brief_score_label *Format the latest bounded research score*

Description

Format the latest bounded research score

Usage

```
brief_score_label(score, moderate = 40, elevated = 70)
```

Arguments

score	Numeric score on a 0 to 100 scale.
moderate	Minimum score labeled moderate.
elevated	Minimum score labeled elevated.

Value

A concise reader-facing character label.

Examples

```
prep_fill_forward(c(NA, 1, NA, 2))
a <- data.frame(date = as.Date("2025-01-01") + c(0, 2),
                value = c(4, 4.2), series = "A")
b <- data.frame(date = as.Date("2025-01-01") + c(1, 2),
                value = c(3, 3.1), series = "B")
prep_series_wide(list(a, b))
prep_zscore(c(1, 2, 3, 4))
prep_recent_changes(c(4, 4.1, 4.2, 4.15), lookback = 2)
score <- factor_bounded_score(prep_zscore(c(1, 2, 3, 4)))
brief_score_label(tail(score, 1))
```

eval_event_performance *Evaluate event performance*

Description

Evaluate event performance

Usage

```
eval_event_performance(
  DT,
  event_col,
  event_score_cols = character(0L),
  H = 1L:42L,
  pos_hit_threshold = 0.05,
  neg_hit_threshold = -0.05
)
```

Arguments

DT	data.table with `datetime`, `close`; optionally `attributes(DT)$inst_id`.
event_col	string; column in `DT` with 1L at event rows.
event_score_cols	character vector of extra columns to carry per case.
H	integer horizons (default `1L:42L`).
pos_hit_threshold	Positive return threshold used to count upside hits.
neg_hit_threshold	Negative return threshold used to count downside hits.

Value

Invisibly: data.table (one row) with summary stats and list-cols `performance_by_cases`, `performance_by_horizons`.

Examples

```
prices <- data.table::data.table(
  datetime = as.POSIXct("2025-01-01", tz = "UTC") + 86400 * 0:59,
  close = 100 * exp(0.001 * (0:59) + 0.01 * sin(0:59)),
  event_a = as.integer(seq_len(60) %% 10 == 0),
  event_b = as.integer(seq_len(60) %% 12 == 0)
)
attr(prices, "inst_id") <- "Synthetic asset"
a <- eval_event_performance(prices, "event_a", H = 1:5)
b <- eval_event_performance(prices, "event_b", H = 1:5)
gen_plot_event_tsline_cum_ret(a)
gen_plot_comparing_events(a, b, "Event A", "Event B")
```

---

eval_portfolio_performance

Evaluate optimized portfolio performance

Description

Evaluate optimized portfolio performance

Usage

```
eval_portfolio_performance(bt_res_list, target_return)
```

Arguments

`bt_res_list`	data.table of backtest rows (with list-col `log_ret_dt`).
`target_return`	numeric annual target return.

Value

Invisibly: single-row data.table (portfolio backtest) with metrics and list-col `log_ret_dt`.

Examples

```
datetime <- as.POSIXct("2025-01-01", tz = "UTC") + 86400 * 0:59
results <- data.table::rbindlist(lapply(1:3, function(i) {
  equity <- exp(cumsum(0.0005 * i + 0.004 * sin((1:60) * i)))
  prep_backtest_result_from_equity(equity, datetime, paste("Asset", i), "Hold")
}))
# Choose a feasible target from the actual synthetic results.
target <- min(results$annual_return)
get_optimal_weights(results, target_return = target)
eval_portfolio_performance(results, target_return = target)
```

`eval_strat_performance` *Evaluate strategy performance*

Description

Evaluate strategy performance

Usage

```
eval_strat_performance(
  DT,
  pos_col_name,
  bg_time = as.POSIXct(NA),
  ed_time = as.POSIXct(NA),
  fee_rate = 7e-04,
  funding_rate = 4e-05,
  interest_rate = 5e-05,
  leverage = 1,
  rf_rate = 0,
  mode = c("new_open", "last_close"),
  fee_sides = 1L,
  tz = Sys.timezone()
)
```

Arguments

DT	data.table with datetime, open, close, and a position column.
pos_col_name	string; name of the position column in DT.
bg_time, ed_time	POSIXct; inclusive start, exclusive end; may be NA.
fee_rate	numeric fee per unit abs(position change).
funding_rate	Legacy funding-rate input retained for compatibility.
interest_rate	Legacy interest-rate input retained for compatibility.
leverage	Legacy leverage input retained for compatibility.
rf_rate	numeric annual risk-free rate.
mode	one of "new_open", "last_close"; execution pricing basis.
fee_sides	Number of sides used when applying the simplified fee model.
tz	Time zone used when coercing input timestamps.

Value

Invisibly: data.table (one row) with metrics and list-col log_ret_dt.

Examples

```
prices <- data.table::data.table(
  datetime = as.POSIXct("2025-01-01", tz = "UTC") + 86400 * 0:39,
  open = 100 + (0:39) / 10,
  close = 100 + (0:39) / 10 + sin(0:39),
  position = rep(c(0, 1), each = 5, length.out = 40)
)
attr(prices, "inst_id") <- "Synthetic asset"
attr(prices$position, "strat_name") <- "Illustration"
attr(prices$position, "strat_par") <- list(window = 5)
result <- eval_strat_performance(prices, "position", tz = "UTC")
eval_strat_plot_tsline_eq(result)
eval_strat_plot_scatter_maxdd_annret(result)

# An external backtest can instead supply an equity curve.
adapted <- prep_backtest_result_from_equity(
  exp(cumsum(0.001 + 0.003 * sin(0:39))), prices$datetime,
  "Synthetic asset", "Illustrative equity curve"
)
adapted[, c("asset_name", "annual_return", "max_drawdown")]
```

eval_strat_plot_scatter_maxdd_annret

Scatter: annual return vs max drawdown

Description

Scatter: annual return vs max drawdown

Usage

```
eval_strat_plot_scatter_maxdd_annret(
  bt_res_list,
  plot_title = "",
  opt_portfolio_line = FALSE,
  style = NULL,
  context = NULL
)
```

Arguments

`bt_res_list`	data.table of backtest rows (from `eval_strat_performance()`).
`plot_title`	string.
`opt_portfolio_line`	logical; unused placeholder.
`style`	Named style family or ‘NULL‘ to use the package default.
`context`	Output context or ‘NULL‘ to use the package default.

Value

ggplot object.

Examples

```
prices <- data.table::data.table(
  datetime = as.POSIXct("2025-01-01", tz = "UTC") + 86400 * 0:39,
  open = 100 + (0:39) / 10,
  close = 100 + (0:39) / 10 + sin(0:39),
  position = rep(c(0, 1), each = 5, length.out = 40)
)
attr(prices, "inst_id") <- "Synthetic asset"
attr(prices$position, "strat_name") <- "Illustration"
attr(prices$position, "strat_par") <- list(window = 5)
result <- eval_strat_performance(prices, "position", tz = "UTC")
eval_strat_plot_tsline_eq(result)
eval_strat_plot_scatter_maxdd_annret(result)

# An external backtest can instead supply an equity curve.
```

```
adapted <- prep_backtest_result_from_equity(
  exp(cumsum(0.001 + 0.003 * sin(0:39))), prices$datetime,
  "Synthetic asset", "Illustrative equity curve"
)
adapted[, c("asset_name", "annual_return", "max_drawdown")]
```

---

eval_strat_plot_tsline_eq

Plot equity curve (time series)

Description

Plot equity curve (time series)

Usage

```
eval_strat_plot_tsline_eq(
  bt_res,
  benchmark_bt_res = NULL,
  bt_res_2 = NULL,
  style = NULL,
  context = NULL,
  caption = NULL,
  log_eq = FALSE
)
```

Arguments

`bt_res`	single-row data.table from `eval_strat_performance()`.
`benchmark_bt_res`	Optional benchmark strategy result to compare against the evaluated strategy.
`bt_res_2`	Optional second evaluated strategy result. Use sparingly; the chart is designed for at most two evaluated strategies plus one benchmark.
`style`	Named style family or ‘NULL‘ to use the package default.
`context`	Output context or ‘NULL‘ to use the package default.
`caption`	Optional chart caption placed above the configured compiler footer.
`log_eq`	Logical; if ‘TRUE‘, plot `log(eq)` on the y-axis instead of raw equity.

Value

ggplot object.

Examples

```
prices <- data.table::data.table(
  datetime = as.POSIXct("2025-01-01", tz = "UTC") + 86400 * 0:39,
  open = 100 + (0:39) / 10,
  close = 100 + (0:39) / 10 + sin(0:39),
  position = rep(c(0, 1), each = 5, length.out = 40)
)
attr(prices, "inst_id") <- "Synthetic asset"
attr(prices$position, "strat_name") <- "Illustration"
attr(prices$position, "strat_par") <- list(window = 5)
result <- eval_strat_performance(prices, "position", tz = "UTC")
eval_strat_plot_tsline_eq(result)
eval_strat_plot_scatter_maxdd_annret(result)

# An external backtest can instead supply an equity curve.
adapted <- prep_backtest_result_from_equity(
  exp(cumsum(0.001 + 0.003 * sin(0:39))), prices$datetime,
  "Synthetic asset", "Illustrative equity curve"
)
adapted[, c("asset_name", "annual_return", "max_drawdown")]
```

---

factor_bounded_score *Map standardized pressure into a bounded research score*

Description

Map standardized pressure into a bounded research score

Usage

```
factor_bounded_score(z, center = 0, scale = 1.2)
```

Arguments

z	Numeric standardized signal.
center	Logistic midpoint.
scale	Positive logistic scale.

Value

Numeric score bounded between 0 and 100.

Examples

```
prep_fill_forward(c(NA, 1, NA, 2))
a <- data.frame(date = as.Date("2025-01-01") + c(0, 2),
                value = c(4, 4.2), series = "A")
b <- data.frame(date = as.Date("2025-01-01") + c(1, 2),
```

```
                 value = c(3, 3.1), series = "B")
prep_series_wide(list(a, b))
prep_zscore(c(1, 2, 3, 4))
prep_recent_changes(c(4, 4.1, 4.2, 4.15), lookback = 2)
score <- factor_bounded_score(prep_zscore(c(1, 2, 3, 4)))
brief_score_label(tail(score, 1))
```

`gen_candle_plots_with_sr_dts`
Generate candlestick plot with support and resistance zones

Description

Generate candlestick plot with support and resistance zones

Usage

```
gen_candle_plots_with_sr_dts(
  DT,
  inst_id,
  bar,
  support_dt = NULL,
  resistance_dt = NULL,
  near_frac = 0.01,
  label_digits = 2,
  show_ema_lines = FALSE,
  style = NULL,
  context = NULL,
  show_compiler = TRUE
)
```

Arguments

`DT`	A data.table containing OHLC data with columns `datetime`, `open`, `high`, `low`, and `close`.
`inst_id`	Instrument label used in the chart title.
`bar`	Bar interval label used in the chart title.
`support_dt`	data.table of support zones with `zone_center`, `zone_low`, and `zone_high`.
`resistance_dt`	data.table of resistance zones with `zone_center`, `zone_low`, and `zone_high`.
`near_frac`	Fraction of the displayed price range used to detect nearby support/resistance labels that should be vertically separated.
`label_digits`	Number of decimals used in support/resistance labels.
`show_ema_lines`	Whether to draw EMA columns when `ema_20`, `ema_50`, `ema_100`, and `ema_200` exist in `DT`.
`style`	Named style family or ‘NULL‘ to use the package default.
`context`	Output context or ‘NULL‘ to use the package default.
`show_compiler`	Whether to append the configured plot compiler name to the plot caption.

Value

A ggplot2 object showing candlesticks with support/resistance zones.

Examples

```
bars <- data.table::data.table(
  datetime = as.POSIXct("2025-01-01", tz = "UTC") + 86400 * 0:9,
  open = 100 + (0:9), close = 100 + (0:9) + sin(0:9)
)
bars[, high := pmax(open, close) + 1]
bars[, low := pmin(open, close) - 1]
bars[, `:=`(atr_logr_12 = 0.025 + 0.002 * sin(0:9),
            atr_q_10_12_300 = 0.015, atr_q_20_12_300 = 0.020,
            atr_q_80_12_300 = 0.030, atr_q_90_12_300 = 0.035)]
viz_candle_base(bars, show_compiler = FALSE)
gen_candle_plots_with_sr_lines(bars, support_pts = 99, resistance_pts = 110)
support <- data.table::data.table(zone_low = 98, zone_high = 100,
                                  zone_center = 99, score = 1)
resistance <- data.table::data.table(zone_low = 109, zone_high = 111,
                                     zone_center = 110, score = 1)
gen_candle_plots_with_sr_dts(bars, "Synthetic asset", "1D",
                             support_dt = support, resistance_dt = resistance,
                             show_compiler = FALSE)
denominator <- data.table::copy(bars)
denominator[, c("open", "high", "low", "close") := lapply(.SD, function(x) x / 2),
            .SDcols = c("open", "high", "low", "close")]
head(prep_ratio_ohlc(bars, denominator, "A/B"))
```

---

gen_candle_plots_with_sr_lines

Generate candlestick plot with support and resistance lines

Description

Generate candlestick plot with support and resistance lines

Usage

```
gen_candle_plots_with_sr_lines(
  DT,
  support_pts,
  resistance_pts,
  near_frac = 0.01,
  label_digits = 2,
  show_ema_lines = FALSE,
  style = NULL,
  context = NULL
)
```

Arguments

DT	A data.table containing OHLC data with columns datetime, open, high, low, and close.
support_pts	Numeric vector of support price levels.
resistance_pts	Numeric vector of resistance price levels.
near_frac	Fraction of the displayed price range used to detect nearby support/resistance labels that should be vertically separated.
label_digits	Number of decimals used in support/resistance labels.
show_ema_lines	Whether to draw EMA columns when ema_20, ema_50, ema_100, and ema_200 exist in DT.
style	Named style family or ‘NULL‘ to use the package default.
context	Output context or ‘NULL‘ to use the package default.

Value

A ggplot2 object showing candlestick chart with support and resistance lines.

Examples

```
bars <- data.table::data.table(
  datetime = as.POSIXct("2025-01-01", tz = "UTC") + 86400 * 0:9,
  open = 100 + (0:9), close = 100 + (0:9) + sin(0:9)
)
bars[, high := pmax(open, close) + 1]
bars[, low := pmin(open, close) - 1]
bars[, `:=`(atr_logr_12 = 0.025 + 0.002 * sin(0:9),
            atr_q_10_12_300 = 0.015, atr_q_20_12_300 = 0.020,
            atr_q_80_12_300 = 0.030, atr_q_90_12_300 = 0.035)]
viz_candle_base(bars, show_compiler = FALSE)
gen_candle_plots_with_sr_lines(bars, support_pts = 99, resistance_pts = 110)
support <- data.table::data.table(zone_low = 98, zone_high = 100,
                                  zone_center = 99, score = 1)
resistance <- data.table::data.table(zone_low = 109, zone_high = 111,
                                     zone_center = 110, score = 1)
gen_candle_plots_with_sr_dts(bars, "Synthetic asset", "1D",
                             support_dt = support, resistance_dt = resistance,
                             show_compiler = FALSE)
denominator <- data.table::copy(bars)
denominator[, c("open", "high", "low", "close") := lapply(.SD, function(x) x / 2),
            .SDcols = c("open", "high", "low", "close")]
head(prep_ratio_ohlc(bars, denominator, "A/B"))
```

gen_facet_plot_from_multicol_ts

Plot multiple time-series columns as facets

Description

Plot multiple time-series columns as facets

Usage

```
gen_facet_plot_from_multicol_ts(
  DT,
  id_vars,
  measure_vars,
  measure_labels,
  style = NULL,
  context = NULL,
  show_compiler = TRUE
)
```

Arguments

DT	A data.table or data.frame containing id and measure columns.
id_vars	Character vector of identifier columns preserved during reshaping.
measure_vars	Character vector of numeric measure columns to plot.
measure_labels	Optional named character vector used to relabel measure_vars in the facet headers.
style	Named style family or ‘NULL‘ to use the package default.
context	Output context or ‘NULL‘ to use the package default.
show_compiler	Whether to append the configured plot compiler name to the plot caption.

Value

A faceted ggplot object.

Examples

```
data <- data.table::data.table(
  datetime = as.POSIXct("2025-01-01", tz = "UTC") + 86400 * 0:9,
  first = 1:10, second = 10:1
)
p <- ggplot2::ggplot(data, ggplot2::aes(datetime, first)) + ggplot2::geom_line()
board <- gen_grid_of_plots_with_labels(list(p, p), n_rows = 1, n_cols = 2,
                                       title = "Two views", show_compiler = FALSE)
grid::grid.newpage()
grid::grid.draw(board)
gen_facet_plot_from_multicol_ts(data, "datetime", c("first", "second"),
                                c("First", "Second"), show_compiler = FALSE)
```

gen_grid_of_plots_with_labels

Arrange plots in a labeled grid layout

Description

Arrange plots in a labeled grid layout

Usage

```
gen_grid_of_plots_with_labels(
  plots,
  n_rows,
  n_cols,
  row_labs = NULL,
  col_labs = NULL,
  title = NULL,
  bottom = NULL,
  row_label_width_cm = 2.2,
  col_header_height_cm = 1.2,
  row_label_gp = NULL,
  col_label_gp = NULL,
  title_gp = NULL,
  bottom_gp = NULL,
  style = NULL,
  context = NULL,
  show_compiler = TRUE
)
```

Arguments

`plots`	List of ggplot objects or grobs.
`n_rows`	Number of plot rows.
`n_cols`	Number of plot columns.
`row_labs`	Optional row labels.
`col_labs`	Optional column labels.
`title`	Optional overall title.
`bottom`	Optional overall footer text.
`row_label_width_cm`	Width allocated to row labels.
`col_header_height_cm`	Height allocated to column headers.
`row_label_gp`	Optional ‘grid::gpar()‘ for row labels.
`col_label_gp`	Optional ‘grid::gpar()‘ for column labels.

title_gp	Optional ‘grid::gpar()‘ for the title.
bottom_gp	Optional ‘grid::gpar()‘ for the footer.
style	Named style family or ‘NULL‘ to use the package default.
context	Output context or ‘NULL‘ to use the package default.
show_compiler	Whether to append the configured plot compiler name to the footer.

Value

A ‘gtable‘ object returned by ‘gridExtra::arrangeGrob()‘.

Examples

```
data <- data.table::data.table(
  datetime = as.POSIXct("2025-01-01", tz = "UTC") + 86400 * 0:9,
  first = 1:10, second = 10:1
)
p <- ggplot2::ggplot(data, ggplot2::aes(datetime, first)) + ggplot2::geom_line()
board <- gen_grid_of_plots_with_labels(list(p, p), n_rows = 1, n_cols = 2,
                                       title = "Two views", show_compiler = FALSE)
grid::grid.newpage()
grid::grid.draw(board)
gen_facet_plot_from_multicol_ts(data, "datetime", c("first", "second"),
                                c("First", "Second"), show_compiler = FALSE)
```

---

gen_plot_comparing_events

Compare two event-study average paths

Description

Compare two event-study average paths

Usage

```
gen_plot_comparing_events(
  first_event_res,
  second_event_res,
  first_event_label,
  second_event_label,
  first_event_color = NULL,
  second_event_color = NULL,
  style = NULL,
  context = NULL
)
```

Arguments

`first_event_res`, `second_event_res`
: Event-performance results returned by `eval_event_performance()`.

`first_event_label`, `second_event_label`
: Labels used in the comparison legend.

`first_event_color`, `second_event_color`
: Optional line colors.

`style`
: Named style family or ‘NULL‘ to use the package default.

`context`
: Output context or ‘NULL‘ to use the package default.

Value

A ggplot object.

Examples

```
prices <- data.table::data.table(
  datetime = as.POSIXct("2025-01-01", tz = "UTC") + 86400 * 0:59,
  close = 100 * exp(0.001 * (0:59) + 0.01 * sin(0:59)),
  event_a = as.integer(seq_len(60) %% 10 == 0),
  event_b = as.integer(seq_len(60) %% 12 == 0)
)
attr(prices, "inst_id") <- "Synthetic asset"
a <- eval_event_performance(prices, "event_a", H = 1:5)
b <- eval_event_performance(prices, "event_b", H = 1:5)
gen_plot_event_tsline_cum_ret(a)
gen_plot_comparing_events(a, b, "Event A", "Event B")
```

---

gen_plot_event_tsline_cum_ret

Plot cumulative event-study return paths

Description

Plot cumulative event-study return paths

Usage

```
gen_plot_event_tsline_cum_ret(
  res,
  threshold = 0.02,
  style = NULL,
  context = NULL
)
```

Arguments

`res`	Event-performance result returned by `eval_event_performance()`.
`threshold`	Positive and negative horizontal reference threshold.
`style`	Named style family or ‘NULL‘ to use the package default.
`context`	Output context or ‘NULL‘ to use the package default.

Value

A ggplot object.

Examples

```
prices <- data.table::data.table(
  datetime = as.POSIXct("2025-01-01", tz = "UTC") + 86400 * 0:59,
  close = 100 * exp(0.001 * (0:59) + 0.01 * sin(0:59)),
  event_a = as.integer(seq_len(60) %% 10 == 0),
  event_b = as.integer(seq_len(60) %% 12 == 0)
)
attr(prices, "inst_id") <- "Synthetic asset"
a <- eval_event_performance(prices, "event_a", H = 1:5)
b <- eval_event_performance(prices, "event_b", H = 1:5)
gen_plot_event_tsline_cum_ret(a)
gen_plot_comparing_events(a, b, "Event A", "Event B")
```

`gen_rolling_monthly_stats` *Generate rolling monthly strategy statistics*

Description

Generate rolling monthly strategy statistics

Usage

```
gen_rolling_monthly_stats(transaction_dt, N_rolling = 6L, debug_mode = FALSE)
```

Arguments

`transaction_dt`	data.table of strategy transactions or per-bar returns.
`N_rolling`	Number of months in the rolling window.
`debug_mode`	Whether to retain additional diagnostic columns.

Value

A data.table of rolling monthly performance statistics.

Examples

```
transactions <- data.table::data.table(
  datetime = as.POSIXct(c("2025-01-15", "2025-02-15", "2025-03-15"), tz = "UTC"),
  action = "close", pre_fee_log_ret = c(0.02, -0.01, 0.03),
  is_win = c(TRUE, FALSE, TRUE)
)
gen_rolling_monthly_stats(transactions, N_rolling = 2)
```

---

gen_yield_curve_plot *Generate U.S. Treasury yield curve plot*

Description

Generate U.S. Treasury yield curve plot

Usage

```
gen_yield_curve_plot(
  DT,
  selected_windows = c("Now", "One week ago", "One month ago"),
  style = NULL,
  context = NULL
)
```

Arguments

DT	A data.table returned from `get_yield_data_DT()`.
selected_windows	Character vector specifying which yield curves to display (e.g., "Now", "One week ago").
style	Named style family or ‘NULL‘ to use the package default.
context	Output context or ‘NULL‘ to use the package default.

Value

A ggplot2 object visualizing selected yield curves.

Examples

```
series <- lapply(seq_len(11), function(i) {
  data.table::data.table(date = as.Date(c("2025-01-01", "2025-02-01")),
                         value = c(4 + i / 20, 3.8 + i / 20))
})
curves <- get_yield_data_DT(series, list(
  Now = as.Date("2025-02-01"), Previous = as.Date("2025-01-01")
))
gen_yield_curve_plot(curves, selected_windows = c("Now", "Previous"))
gen_yield_curve_plot_grid(curves, selected_windows = c("Now", "Previous"))
```

gen_yield_curve_plot_grid

Generate yield curve comparison grid

Description

Generate yield curve comparison grid

Usage

```
gen_yield_curve_plot_grid(
  DT,
  selected_windows = c("Now", "One week ago", "One month ago", "Six months ago",
    "One year ago"),
  style = NULL,
  context = NULL
)
```

Arguments

DT	A data.table returned from `get_yield_data_DT()`.
selected_windows	Character vector specifying yield curve snapshots for comparison.
style	Named style family or ‘NULL‘ to use the package default.
context	Output context or ‘NULL‘ to use the package default.

Value

A faceted ggplot2 object comparing yield curves across multiple time windows.

Examples

```
series <- lapply(seq_len(11), function(i) {
  data.table::data.table(date = as.Date(c("2025-01-01", "2025-02-01")),
                         value = c(4 + i / 20, 3.8 + i / 20))
})
curves <- get_yield_data_DT(series, list(
  Now = as.Date("2025-02-01"), Previous = as.Date("2025-01-01")
))
gen_yield_curve_plot(curves, selected_windows = c("Now", "Previous"))
gen_yield_curve_plot_grid(curves, selected_windows = c("Now", "Previous"))
```

get_optimal_weights *Compute optimal strategy weights*

Description

Compute optimal strategy weights

Usage

```
get_optimal_weights(bt_res_list, target_return = 0.1)
```

Arguments

bt_res_list	data.table of backtest rows (from eval_strat_performance()).
target_return	numeric annual target return (e.g., 0.10).

Value

List with optimal_weights (named numeric) and optimal_weights_table (data.frame of nonzero weights).

Examples

```
datetime <- as.POSIXct("2025-01-01", tz = "UTC") + 86400 * 0:59
results <- data.table::rbindlist(lapply(1:3, function(i) {
  equity <- exp(cumsum(0.0005 * i + 0.004 * sin((1:60) * i)))
  prep_backtest_result_from_equity(equity, datetime, paste("Asset", i), "Hold")
}))
# Choose a feasible target from the actual synthetic results.
target <- min(results$annual_return)
get_optimal_weights(results, target_return = target)
eval_portfolio_performance(results, target_return = target)
```

get_yield_data_DT *Get yield data for given dates*

Description

Get yield data for given dates

Usage

```
get_yield_data_DT(yield_dt_list, yield_dates)
```

Arguments

`yield_dt_list`	A list of data.tables, each containing a yield series with columns `date` and `value`.
`yield_dates`	A vector of dates for which yields should be retrieved.

Value

A data.table containing yield data across maturities and dates.

Examples

```
series <- lapply(seq_len(11), function(i) {
  data.table::data.table(date = as.Date(c("2025-01-01", "2025-02-01")),
                         value = c(4 + i / 20, 3.8 + i / 20))
})
curves <- get_yield_data_DT(series, list(
  Now = as.Date("2025-02-01"), Previous = as.Date("2025-01-01")
))
gen_yield_curve_plot(curves, selected_windows = c("Now", "Previous"))
gen_yield_curve_plot_grid(curves, selected_windows = c("Now", "Previous"))
```

`prep_backtest_result_from_equity` *Build A Backtest Result Row From An Equity Curve*

Description

Converts an externally generated equity curve into the compact backtest result shape used by investlabr plotting helpers.

Usage

```
prep_backtest_result_from_equity(equity, datetime, asset_name, strat_label)
```

Arguments

`equity`	Numeric equity curve.
`datetime`	POSIXct-compatible vector aligned with ‘equity‘.
`asset_name`	Character asset label.
`strat_label`	Character strategy label.

Value

A one-row ‘data.table‘ with summary metrics and a ‘log_ret_dt‘ list-column.

Examples

```
prices <- data.table::data.table(
  datetime = as.POSIXct("2025-01-01", tz = "UTC") + 86400 * 0:39,
  open = 100 + (0:39) / 10,
  close = 100 + (0:39) / 10 + sin(0:39),
  position = rep(c(0, 1), each = 5, length.out = 40)
)
attr(prices, "inst_id") <- "Synthetic asset"
attr(prices$position, "strat_name") <- "Illustration"
attr(prices$position, "strat_par") <- list(window = 5)
result <- eval_strat_performance(prices, "position", tz = "UTC")
eval_strat_plot_tsline_eq(result)
eval_strat_plot_scatter_maxdd_annret(result)

# An external backtest can instead supply an equity curve.
adapted <- prep_backtest_result_from_equity(
  exp(cumsum(0.001 + 0.003 * sin(0:39))), prices$datetime,
  "Synthetic asset", "Illustrative equity curve"
)
adapted[, c("asset_name", "annual_return", "max_drawdown")]
```

---

prep_fill_forward *Carry the latest observed value forward*

Description

Carry the latest observed value forward

Usage

```
prep_fill_forward(x)
```

Arguments

x	Vector containing values and missing observations.

Value

A vector with missing values filled from the latest prior observation.

Examples

```
prep_fill_forward(c(NA, 1, NA, 2))
a <- data.frame(date = as.Date("2025-01-01") + c(0, 2),
                value = c(4, 4.2), series = "A")
b <- data.frame(date = as.Date("2025-01-01") + c(1, 2),
                value = c(3, 3.1), series = "B")
prep_series_wide(list(a, b))
prep_zscore(c(1, 2, 3, 4))
```

```
prep_recent_changes(c(4, 4.1, 4.2, 4.15), lookback = 2)
score <- factor_bounded_score(prep_zscore(c(1, 2, 3, 4)))
brief_score_label(tail(score, 1))
```

`prep_ratio_ohlc` *Build Synthetic OHLC For A Price Ratio*

Description

Builds a close-aligned OHLC proxy for a ratio between two OHLC series. This is intended for research-side spread or curve proxy visualization, not production execution.

Usage

```
prep_ratio_ohlc(numerator_dt, denominator_dt, ratio_symbol)
```

Arguments

`numerator_dt`, `denominator_dt`
: ‘data.table‘-like OHLC inputs containing ‘datetime‘, ‘open‘, ‘high‘, ‘low‘, ‘close‘, and optionally ‘volume‘.

`ratio_symbol`
: Character label for the synthetic ratio.

Value

A ‘data.table‘ with OHLC columns for the synthetic ratio.

Examples

```
bars <- data.table::data.table(
  datetime = as.POSIXct("2025-01-01", tz = "UTC") + 86400 * 0:9,
  open = 100 + (0:9), close = 100 + (0:9) + sin(0:9)
)
bars[, high := pmax(open, close) + 1]
bars[, low := pmin(open, close) - 1]
bars[, `:=`(atr_logr_12 = 0.025 + 0.002 * sin(0:9),
            atr_q_10_12_300 = 0.015, atr_q_20_12_300 = 0.020,
            atr_q_80_12_300 = 0.030, atr_q_90_12_300 = 0.035)]
viz_candle_base(bars, show_compiler = FALSE)
gen_candle_plots_with_sr_lines(bars, support_pts = 99, resistance_pts = 110)
support <- data.table::data.table(zone_low = 98, zone_high = 100,
                                  zone_center = 99, score = 1)
resistance <- data.table::data.table(zone_low = 109, zone_high = 111,
                                     zone_center = 110, score = 1)
gen_candle_plots_with_sr_dts(bars, "Synthetic asset", "1D",
                             support_dt = support, resistance_dt = resistance,
                             show_compiler = FALSE)
denominator <- data.table::copy(bars)
denominator[, c("open", "high", "low", "close") := lapply(.SD, function(x) x / 2),
            .SDcols = c("open", "high", "low", "close")]
head(prep_ratio_ohlc(bars, denominator, "A/B"))
```

prep_recent_changes *Extract recent finite first differences*

Description

Extract recent finite first differences

Usage

```
prep_recent_changes(x, lookback = 252L)
```

Arguments

x	Numeric level series.
lookback	Maximum number of recent changes to return.

Value

Numeric vector of recent finite first differences.

Examples

```
prep_fill_forward(c(NA, 1, NA, 2))
a <- data.frame(date = as.Date("2025-01-01") + c(0, 2),
                value = c(4, 4.2), series = "A")
b <- data.frame(date = as.Date("2025-01-01") + c(1, 2),
                value = c(3, 3.1), series = "B")
prep_series_wide(list(a, b))
prep_zscore(c(1, 2, 3, 4))
prep_recent_changes(c(4, 4.1, 4.2, 4.15), lookback = 2)
score <- factor_bounded_score(prep_zscore(c(1, 2, 3, 4)))
brief_score_label(tail(score, 1))
```

prep_series_wide *Combine labeled time series into a research-ready wide table*

Description

Combine labeled time series into a research-ready wide table

Usage

```
prep_series_wide(series_list, fill_forward = TRUE)
```

Arguments

series_list	List of data frames or data.tables containing date, value, and series columns.
fill_forward	Whether to carry each series forward after reshaping.

Value

A data.table with one row per date and one column per series.

Examples

```
prep_fill_forward(c(NA, 1, NA, 2))
a <- data.frame(date = as.Date("2025-01-01") + c(0, 2),
                value = c(4, 4.2), series = "A")
b <- data.frame(date = as.Date("2025-01-01") + c(1, 2),
                value = c(3, 3.1), series = "B")
prep_series_wide(list(a, b))
prep_zscore(c(1, 2, 3, 4))
prep_recent_changes(c(4, 4.1, 4.2, 4.15), lookback = 2)
score <- factor_bounded_score(prep_zscore(c(1, 2, 3, 4)))
brief_score_label(tail(score, 1))
```

---

prep_zscore *Standardize a numeric series*

Description

Standardize a numeric series

Usage

```
prep_zscore(x)
```

Arguments

x	Numeric vector.

Value

Numeric z-scores. Returns all NA when dispersion is zero or unavailable.

Examples

```
prep_fill_forward(c(NA, 1, NA, 2))
a <- data.frame(date = as.Date("2025-01-01") + c(0, 2),
                value = c(4, 4.2), series = "A")
b <- data.frame(date = as.Date("2025-01-01") + c(1, 2),
                value = c(3, 3.1), series = "B")
prep_series_wide(list(a, b))
prep_zscore(c(1, 2, 3, 4))
```

```
prep_recent_changes(c(4, 4.1, 4.2, 4.15), lookback = 2)
score <- factor_bounded_score(prep_zscore(c(1, 2, 3, 4)))
brief_score_label(tail(score, 1))
```

`sim_ar1_dgp` *Simulate an AR(1) data-generating process*

Description

Simulate an AR(1) data-generating process

Usage

```
sim_ar1_dgp(
  n,
  a1,
  y_lag1 = 0,
  burn = 50,
  sigma = 1,
  drift = 0,
  trend_slope = 0,
  seed = NULL
)
```

Arguments

`n`	Number of observations returned after burn-in.
`a1`	AR(1) coefficient.
`y_lag1`	Initial lagged value.
`burn`	Number of burn-in observations discarded before returning the sample.
`sigma`	Innovation standard deviation.
`drift`	Constant drift term added each step.
`trend_slope`	Linear trend slope applied to the running time index.
`seed`	Optional integer seed for reproducibility.

Value

‘data.table‘ with columns ‘t‘, ‘value‘, and ‘innovation‘.

Examples

```
head(sim_ar1_dgp(30, a1 = 0.7, seed = 1))
head(sim_ar2_dgp(30, a1 = 0.5, a2 = 0.2, seed = 1))
head(sim_ar3_dgp(30, a1 = 0.4, a2 = 0.2, a3 = 0.1, seed = 1))
head(sim_ma1_dgp(30, b1 = 0.5, seed = 1))
head(sim_arch1_dgp(30, alpha0 = 0.1, alpha1 = 0.4, seed = 1))
head(sim_garch11_dgp(30, alpha0 = 0.1, alpha1 = 0.1, beta1 = 0.8, seed = 1))
```

`sim_ar2_dgp` *Simulate an AR(2) data-generating process*

Description

Simulate an AR(2) data-generating process

Usage

```
sim_ar2_dgp(
  n,
  a1,
  a2,
  y_lag1 = 0,
  y_lag2 = 0,
  burn = 50,
  sigma = 1,
  drift = 0,
  trend_slope = 0,
  seed = NULL
)
```

Arguments

`n`	Number of observations returned after burn-in.
`a1`	AR(1) coefficient.
`a2`	AR(2) coefficient.
`y_lag1`	Initial lagged value.
`y_lag2`	Second initial lagged value.
`burn`	Number of burn-in observations discarded before returning the sample.
`sigma`	Innovation standard deviation.
`drift`	Constant drift term added each step.
`trend_slope`	Linear trend slope applied to the running time index.
`seed`	Optional integer seed for reproducibility.

Value

‘data.table‘ with columns ‘t‘, ‘value‘, and ‘innovation‘.

Examples

```
head(sim_ar1_dgp(30, a1 = 0.7, seed = 1))
head(sim_ar2_dgp(30, a1 = 0.5, a2 = 0.2, seed = 1))
head(sim_ar3_dgp(30, a1 = 0.4, a2 = 0.2, a3 = 0.1, seed = 1))
head(sim_ma1_dgp(30, b1 = 0.5, seed = 1))
head(sim_arch1_dgp(30, alpha0 = 0.1, alpha1 = 0.4, seed = 1))
head(sim_garch11_dgp(30, alpha0 = 0.1, alpha1 = 0.1, beta1 = 0.8, seed = 1))
```

`sim_ar3_dgp` *Simulate an AR(3) data-generating process*

Description

Simulate an AR(3) data-generating process

Usage

```
sim_ar3_dgp(
  n,
  a1,
  a2,
  a3,
  y_lag1 = 0,
  y_lag2 = 0,
  y_lag3 = 0,
  burn = 50,
  sigma = 1,
  drift = 0,
  trend_slope = 0,
  seed = NULL
)
```

Arguments

`n`	Number of observations returned after burn-in.
`a1`	AR(1) coefficient.
`a2`	AR(2) coefficient.
`a3`	AR(3) coefficient.
`y_lag1`	Initial lagged value.
`y_lag2`	Second initial lagged value.
`y_lag3`	Third initial lagged value.
`burn`	Number of burn-in observations discarded before returning the sample.
`sigma`	Innovation standard deviation.
`drift`	Constant drift term added each step.
`trend_slope`	Linear trend slope applied to the running time index.
`seed`	Optional integer seed for reproducibility.

Value

‘data.table‘ with columns ‘t‘, ‘value‘, and ‘innovation‘.

Examples

```
head(sim_ar1_dgp(30, a1 = 0.7, seed = 1))
head(sim_ar2_dgp(30, a1 = 0.5, a2 = 0.2, seed = 1))
head(sim_ar3_dgp(30, a1 = 0.4, a2 = 0.2, a3 = 0.1, seed = 1))
head(sim_ma1_dgp(30, b1 = 0.5, seed = 1))
head(sim_arch1_dgp(30, alpha0 = 0.1, alpha1 = 0.4, seed = 1))
head(sim_garch11_dgp(30, alpha0 = 0.1, alpha1 = 0.1, beta1 = 0.8, seed = 1))
```

---

sim_arch1_dgp *Simulate an ARCH(1) data-generating process*

Description

Simulate an ARCH(1) data-generating process

Usage

```
sim_arch1_dgp(n, alpha0, alpha1, e_lag1_sq = 0, burn = 50, seed = NULL)
```

Arguments

n	Number of observations returned after burn-in.
alpha0	Constant term in the conditional variance recursion.
alpha1	ARCH coefficient on the lagged squared innovation.
e_lag1_sq	Initial lagged squared innovation.
burn	Number of burn-in observations discarded before returning the sample.
seed	Optional integer seed for reproducibility.

Value

‘data.table‘ with columns ‘t‘, ‘value‘, ‘innovation‘, and ‘sigma2‘.

Examples

```
head(sim_ar1_dgp(30, a1 = 0.7, seed = 1))
head(sim_ar2_dgp(30, a1 = 0.5, a2 = 0.2, seed = 1))
head(sim_ar3_dgp(30, a1 = 0.4, a2 = 0.2, a3 = 0.1, seed = 1))
head(sim_ma1_dgp(30, b1 = 0.5, seed = 1))
head(sim_arch1_dgp(30, alpha0 = 0.1, alpha1 = 0.4, seed = 1))
head(sim_garch11_dgp(30, alpha0 = 0.1, alpha1 = 0.1, beta1 = 0.8, seed = 1))
```

sim_forward_fan — *Simulate a forward percentile fan from historical changes*

Description

Simulate a forward percentile fan from historical changes

Usage

```
sim_forward_fan(
  start_value,
  changes,
  horizons = c(1L, 5L, 10L, 20L, 60L),
  n_paths = 600L,
  seed = 1L
)
```

Arguments

start_value	Current level from which paths begin.
changes	Historical one-period changes sampled with replacement.
horizons	Positive integer horizons included in the returned fan.
n_paths	Number of simulated paths.
seed	Optional reproducibility seed.

Value

A data.table containing p10, p25, p50, p75, and p90 levels by horizon, including horizon zero.

Examples

```
# Fully synthetic inputs; no data service or local cache is needed.
changes <- seq(-0.10, 0.10, length.out = 40)
fan <- sim_forward_fan(4, changes, horizons = c(1, 5, 10),
                       n_paths = 100, seed = 7)
viz_forward_fan(fan, "Illustrative yield paths", "Historical-change bootstrap",
                "Yield (%)", show_compiler = FALSE)
paths <- sim_scenario_path(4, c(Easing = -0.25, Hold = 0, Tightening = 0.25),
                           horizon_months = 3, start_date = as.Date("2025-01-01"))
viz_series_lines(paths, "date", "value", color = "scenario",
                 title = "Policy scenarios", subtitle = "Explicit monthly assumptions",
                 y_label = "Rate (%)", show_compiler = FALSE)
```

sim_garch11_dgp *Simulate a GARCH(1,1) data-generating process*

Description

Simulate a GARCH(1,1) data-generating process

Usage

```
sim_garch11_dgp(
  n,
  alpha0,
  alpha1,
  beta1,
  sigma2_0 = NULL,
  burn = 50,
  z_dist = c("norm", "stdt"),
  df = 8,
  seed = NULL
)
```

Arguments

n	Number of observations returned after burn-in.
alpha0	Constant term in the conditional variance recursion.
alpha1	Coefficient on the lagged squared innovation.
beta1	Coefficient on the lagged conditional variance.
sigma2_0	Optional initial conditional variance. If ‘NULL‘, the unconditional variance is used.
burn	Number of burn-in observations discarded before returning the sample.
z_dist	Innovation distribution: ‘"norm"‘ or ‘"stdt"‘.
df	Degrees of freedom for the standardized Student t innovations.
seed	Optional integer seed for reproducibility.

Value

‘data.table‘ with columns ‘t‘, ‘value‘, ‘innovation‘, and ‘sigma2‘.

Examples

```
head(sim_ar1_dgp(30, a1 = 0.7, seed = 1))
head(sim_ar2_dgp(30, a1 = 0.5, a2 = 0.2, seed = 1))
head(sim_ar3_dgp(30, a1 = 0.4, a2 = 0.2, a3 = 0.1, seed = 1))
head(sim_ma1_dgp(30, b1 = 0.5, seed = 1))
head(sim_arch1_dgp(30, alpha0 = 0.1, alpha1 = 0.4, seed = 1))
head(sim_garch11_dgp(30, alpha0 = 0.1, alpha1 = 0.1, beta1 = 0.8, seed = 1))
```

sim_ma1_dgp — *Simulate an MA(1) data-generating process*

Description

Simulate an MA(1) data-generating process

Usage

```
sim_ma1_dgp(n, b1, e_lag1 = 0, burn = 50, sigma = 1, seed = NULL)
```

Arguments

n	Number of observations returned after burn-in.
b1	MA(1) coefficient.
e_lag1	Initial lagged innovation.
burn	Number of burn-in observations discarded before returning the sample.
sigma	Innovation standard deviation.
seed	Optional integer seed for reproducibility.

Value

‘data.table‘ with columns ‘t‘, ‘value‘, and ‘innovation‘.

Examples

```
head(sim_ar1_dgp(30, a1 = 0.7, seed = 1))
head(sim_ar2_dgp(30, a1 = 0.5, a2 = 0.2, seed = 1))
head(sim_ar3_dgp(30, a1 = 0.4, a2 = 0.2, a3 = 0.1, seed = 1))
head(sim_ma1_dgp(30, b1 = 0.5, seed = 1))
head(sim_arch1_dgp(30, alpha0 = 0.1, alpha1 = 0.4, seed = 1))
head(sim_garch11_dgp(30, alpha0 = 0.1, alpha1 = 0.1, beta1 = 0.8, seed = 1))
```

sim_scenario_path — *Construct deterministic scenario paths*

Description

Construct deterministic scenario paths

Usage

```
sim_scenario_path(
  start_value,
  scenario_changes,
  horizon_months = 6L,
  start_date = Sys.Date()
)
```

Arguments

`start_value`	Current level.
`scenario_changes`	Named numeric vector or list of per-period changes.
`horizon_months`	Number of forward monthly steps.
`start_date`	First scenario date.

Value

Long-form data.table with `date`, `scenario`, and `value` columns.

Examples

```
# Fully synthetic inputs; no data service or local cache is needed.
changes <- seq(-0.10, 0.10, length.out = 40)
fan <- sim_forward_fan(4, changes, horizons = c(1, 5, 10),
                       n_paths = 100, seed = 7)
viz_forward_fan(fan, "Illustrative yield paths", "Historical-change bootstrap",
                "Yield (%)", show_compiler = FALSE)
paths <- sim_scenario_path(4, c(Easing = -0.25, Hold = 0, Tightening = 0.25),
                           horizon_months = 3, start_date = as.Date("2025-01-01"))
viz_series_lines(paths, "date", "value", color = "scenario",
                 title = "Policy scenarios", subtitle = "Explicit monthly assumptions",
                 y_label = "Rate (%)", show_compiler = FALSE)
```

`viz_annotate_event_lines` *Add event-line annotations to a plot*

Description

Add event-line annotations to a plot

Usage

```
viz_annotate_event_lines(
  plot,
  event_dt,
  x_col = "datetime",
  label_col = NULL,
  color = NULL,
  linetype = "dashed",
  alpha = 0.45,
  style = NULL,
  context = NULL
)
```

Arguments

plot	A ‘ggplot‘ object.
event_dt	Data with at least an x-position column.
x_col	Name of the x-position column.
label_col	Optional label column.
color	Optional line and label color.
linetype	Line type for event markers.
alpha	Alpha applied to event lines.
style	Named style family or ‘NULL‘ to use the package default.
context	Output context or ‘NULL‘ to use the package default.

Value

A ‘ggplot‘ object with event annotations.

Examples

```
viz_style_get("policy_memo", "report")
viz_palette_get("institutional_blue")
local({
 old <- options(investlabr.viz.style = "research_note", investlabr.viz.context = "report")
  on.exit(options(old))
  viz_style_set("research_note", "report")
})
data <- data.frame(date = as.Date("2025-01-01") + 0:9, value = 1:10)
p <- ggplot2::ggplot(data, ggplot2::aes(date, value)) + ggplot2::geom_line()
p <- viz_theme_apply(p, style = "policy_memo", show_compiler = FALSE)
p <- viz_annotate_last_value(p, data, "date", "value")
p <- viz_annotate_event_lines(p, data.frame(datetime = as.Date("2025-01-05")))
p <- viz_annotate_regime_bands(p, data.frame(
  start = as.Date("2025-01-02"), end = as.Date("2025-01-04")
))
viz_annotate_source_caption(p, "Synthetic data", "Illustration only")
```

viz_annotate_last_value

Add a direct label for the last observation in a series

Description

Add a direct label for the last observation in a series

Usage

```
viz_annotate_last_value(
  plot,
  data,
  x,
  y,
  label = NULL,
  prefix = "",
  digits = 2,
  color = NULL,
  style = NULL,
  context = NULL
)
```

Arguments

plot	A ‘ggplot‘ object.
data	A data.frame or data.table containing the plotted series.
x	Name of the x column.
y	Name of the y column.
label	Optional name of a label column. If ‘NULL‘, the formatted ‘y‘ value is used.
prefix	Optional prefix prepended to the label text.
digits	Number of digits used when formatting numeric labels.
color	Optional label color.
style	Named style family or ‘NULL‘ to use the package default.
context	Output context or ‘NULL‘ to use the package default.

Value

A ‘ggplot‘ object with a last-value annotation.

Examples

```
viz_style_get("policy_memo", "report")
viz_palette_get("institutional_blue")
local({
 old <- options(investlabr.viz.style = "research_note", investlabr.viz.context = "report")
  on.exit(options(old))
  viz_style_set("research_note", "report")
})
data <- data.frame(date = as.Date("2025-01-01") + 0:9, value = 1:10)
p <- ggplot2::ggplot(data, ggplot2::aes(date, value)) + ggplot2::geom_line()
p <- viz_theme_apply(p, style = "policy_memo", show_compiler = FALSE)
p <- viz_annotate_last_value(p, data, "date", "value")
p <- viz_annotate_event_lines(p, data.frame(datetime = as.Date("2025-01-05")))
p <- viz_annotate_regime_bands(p, data.frame(
  start = as.Date("2025-01-02"), end = as.Date("2025-01-04")
))
viz_annotate_source_caption(p, "Synthetic data", "Illustration only")
```

---

viz_annotate_regime_bands

Add regime-band annotations to a plot

Description

Add regime-band annotations to a plot

Usage

```
viz_annotate_regime_bands(
  plot,
  regime_dt,
  xmin_col = "start",
  xmax_col = "end",
  fill = NULL,
  alpha = 0.12,
  style = NULL,
  context = NULL
)
```

Arguments

plot	A ‘ggplot‘ object.
regime_dt	Data with start and end columns.
xmin_col	Name of the regime start column.
xmax_col	Name of the regime end column.
fill	Optional fill color.

alpha	Optional fill alpha.
style	Named style family or ‘NULL‘ to use the package default.
context	Output context or ‘NULL‘ to use the package default.

Value

A ‘ggplot‘ object with shaded regime bands.

Examples

```
viz_style_get("policy_memo", "report")
viz_palette_get("institutional_blue")
local({
 old <- options(investlabr.viz.style = "research_note", investlabr.viz.context = "report")
  on.exit(options(old))
  viz_style_set("research_note", "report")
})
data <- data.frame(date = as.Date("2025-01-01") + 0:9, value = 1:10)
p <- ggplot2::ggplot(data, ggplot2::aes(date, value)) + ggplot2::geom_line()
p <- viz_theme_apply(p, style = "policy_memo", show_compiler = FALSE)
p <- viz_annotate_last_value(p, data, "date", "value")
p <- viz_annotate_event_lines(p, data.frame(datetime = as.Date("2025-01-05")))
p <- viz_annotate_regime_bands(p, data.frame(
  start = as.Date("2025-01-02"), end = as.Date("2025-01-04")
))
viz_annotate_source_caption(p, "Synthetic data", "Illustration only")
```

---

viz_annotate_source_caption

Append a standardized source caption to a plot

Description

Append a standardized source caption to a plot

Usage

```
viz_annotate_source_caption(plot, source_text, note = NULL)
```

Arguments

plot	A ‘ggplot‘ object.
source_text	Source text.
note	Optional note appended after the source.

Value

A ‘ggplot‘ object with an updated caption.

Examples

```
viz_style_get("policy_memo", "report")
viz_palette_get("institutional_blue")
local({
 old <- options(investlabr.viz.style = "research_note", investlabr.viz.context = "report")
  on.exit(options(old))
  viz_style_set("research_note", "report")
})
data <- data.frame(date = as.Date("2025-01-01") + 0:9, value = 1:10)
p <- ggplot2::ggplot(data, ggplot2::aes(date, value)) + ggplot2::geom_line()
p <- viz_theme_apply(p, style = "policy_memo", show_compiler = FALSE)
p <- viz_annotate_last_value(p, data, "date", "value")
p <- viz_annotate_event_lines(p, data.frame(datetime = as.Date("2025-01-05")))
p <- viz_annotate_regime_bands(p, data.frame(
  start = as.Date("2025-01-02"), end = as.Date("2025-01-04")
))
viz_annotate_source_caption(p, "Synthetic data", "Illustration only")
```

---

viz_candle_base *Generate a base candlestick plot*

Description

Generate a base candlestick plot

Usage

```
viz_candle_base(DT, style = NULL, context = NULL, show_compiler = TRUE)
```

Arguments

DT	A data.table containing OHLC data with columns datetime, open, high, low, and close.
style	Named style family or ‘NULL‘ to use the package default.
context	Output context or ‘NULL‘ to use the package default.
show_compiler	Whether to append the configured plot compiler name to the plot caption.

Value

A ggplot2 object showing an OHLC candlestick chart.

Examples

```
bars <- data.table::data.table(
  datetime = as.POSIXct("2025-01-01", tz = "UTC") + 86400 * 0:9,
  open = 100 + (0:9), close = 100 + (0:9) + sin(0:9)
)
bars[, high := pmax(open, close) + 1]
```

```
bars[, low := pmin(open, close) - 1]
bars[, `:=`(atr_logr_12 = 0.025 + 0.002 * sin(0:9),
            atr_q_10_12_300 = 0.015, atr_q_20_12_300 = 0.020,
            atr_q_80_12_300 = 0.030, atr_q_90_12_300 = 0.035)]
viz_candle_base(bars, show_compiler = FALSE)
gen_candle_plots_with_sr_lines(bars, support_pts = 99, resistance_pts = 110)
support <- data.table::data.table(zone_low = 98, zone_high = 100,
                                  zone_center = 99, score = 1)
resistance <- data.table::data.table(zone_low = 109, zone_high = 111,
                                     zone_center = 110, score = 1)
gen_candle_plots_with_sr_dts(bars, "Synthetic asset", "1D",
                             support_dt = support, resistance_dt = resistance,
                             show_compiler = FALSE)
denominator <- data.table::copy(bars)
denominator[, c("open", "high", "low", "close") := lapply(.SD, function(x) x / 2),
            .SDcols = c("open", "high", "low", "close")]
head(prep_ratio_ohlc(bars, denominator, "A/B"))
```

`viz_forward_fan` *Plot a forward percentile fan*

Description

Plot a forward percentile fan

Usage

```
viz_forward_fan(
  fan_dt,
  title,
  subtitle,
  y_label,
  accent = NULL,
  style = NULL,
  context = NULL,
  show_compiler = TRUE
)
```

Arguments

`fan_dt`	Data frame containing `horizon`, `p10`, `p25`, `p50`, and `p75`, and `p90`.
`title`	Plot title.
`subtitle`	Plot subtitle.
`y_label`	Y-axis label.
`accent`	Optional fan color.
`style`	Named style family or ‘NULL‘ to use the package default.
`context`	Output context or ‘NULL‘ to use the package default.
`show_compiler`	Whether to append the configured compiler footer.

Value

A ggplot object.

Examples

```
# Fully synthetic inputs; no data service or local cache is needed.
changes <- seq(-0.10, 0.10, length.out = 40)
fan <- sim_forward_fan(4, changes, horizons = c(1, 5, 10),
                       n_paths = 100, seed = 7)
viz_forward_fan(fan, "Illustrative yield paths", "Historical-change bootstrap",
                "Yield (%)", show_compiler = FALSE)
paths <- sim_scenario_path(4, c(Easing = -0.25, Hold = 0, Tightening = 0.25),
                           horizon_months = 3, start_date = as.Date("2025-01-01"))
viz_series_lines(paths, "date", "value", color = "scenario",
                 title = "Policy scenarios", subtitle = "Explicit monthly assumptions",
                 y_label = "Rate (%)", show_compiler = FALSE)
```

---

viz_palette_get *Get visualization palette values*

Description

Get visualization palette values

Usage

```
viz_palette_get(
  style = NULL,
  context = NULL,
  palette = c("discrete", "sequential")
)
```

Arguments

style	Named style family or ‘NULL‘ to use the package default.
context	Output context or ‘NULL‘ to use the package default.
palette	One of ‘"discrete"‘ or ‘"sequential"‘.

Value

Character vector of colors.

Examples

```
viz_style_get("policy_memo", "report")
viz_palette_get("institutional_blue")
local({
 old <- options(investlabr.viz.style = "research_note", investlabr.viz.context = "report")
  on.exit(options(old))
  viz_style_set("research_note", "report")
})
data <- data.frame(date = as.Date("2025-01-01") + 0:9, value = 1:10)
p <- ggplot2::ggplot(data, ggplot2::aes(date, value)) + ggplot2::geom_line()
p <- viz_theme_apply(p, style = "policy_memo", show_compiler = FALSE)
p <- viz_annotate_last_value(p, data, "date", "value")
p <- viz_annotate_event_lines(p, data.frame(datetime = as.Date("2025-01-05")))
p <- viz_annotate_regime_bands(p, data.frame(
  start = as.Date("2025-01-02"), end = as.Date("2025-01-04")
))
viz_annotate_source_caption(p, "Synthetic data", "Illustration only")
```

---

viz_series_lines *Plot one or more research series*

Description

Plot one or more research series

Usage

```
viz_series_lines(
  data,
  x,
  y,
  color = "series",
  title,
  subtitle,
  y_label,
  colors = NULL,
  style = NULL,
  context = NULL,
  legend_position = "bottom",
  show_compiler = TRUE
)
```

Arguments

data	Data frame containing the plotted columns.
x, y, color	Column names mapped to x, y, and color.
title	Plot title.

subtitle	Plot subtitle.
y_label	Y-axis label.
colors	Optional named manual color vector.
style	Named style family or ‘NULL‘ to use the package default.
context	Output context or ‘NULL‘ to use the package default.
legend_position	Legend position.
show_compiler	Whether to append the configured compiler footer.

Value

A ggplot object.

Examples

```
# Fully synthetic inputs; no data service or local cache is needed.
changes <- seq(-0.10, 0.10, length.out = 40)
fan <- sim_forward_fan(4, changes, horizons = c(1, 5, 10),
                       n_paths = 100, seed = 7)
viz_forward_fan(fan, "Illustrative yield paths", "Historical-change bootstrap",
                "Yield (%)", show_compiler = FALSE)
paths <- sim_scenario_path(4, c(Easing = -0.25, Hold = 0, Tightening = 0.25),
                           horizon_months = 3, start_date = as.Date("2025-01-01"))
viz_series_lines(paths, "date", "value", color = "scenario",
                 title = "Policy scenarios", subtitle = "Explicit monthly assumptions",
                 y_label = "Rate (%)", show_compiler = FALSE)
```

---

viz_style_get *Get resolved visualization style*

Description

Get resolved visualization style

Usage

```
viz_style_get(style = NULL, context = NULL)
```

Arguments

style	Named style family or ‘NULL‘ to use the package default.
context	Output context or ‘NULL‘ to use the package default.

Value

Named list describing colors, sizing, and theme settings.

Examples

```
viz_style_get("policy_memo", "report")
viz_palette_get("institutional_blue")
local({
 old <- options(investlabr.viz.style = "research_note", investlabr.viz.context = "report")
  on.exit(options(old))
  viz_style_set("research_note", "report")
})
data <- data.frame(date = as.Date("2025-01-01") + 0:9, value = 1:10)
p <- ggplot2::ggplot(data, ggplot2::aes(date, value)) + ggplot2::geom_line()
p <- viz_theme_apply(p, style = "policy_memo", show_compiler = FALSE)
p <- viz_annotate_last_value(p, data, "date", "value")
p <- viz_annotate_event_lines(p, data.frame(datetime = as.Date("2025-01-05")))
p <- viz_annotate_regime_bands(p, data.frame(
  start = as.Date("2025-01-02"), end = as.Date("2025-01-04")
))
viz_annotate_source_caption(p, "Synthetic data", "Illustration only")
```

`viz_style_set` *Set default visualization style and context*

Description

Set default visualization style and context

Usage

```
viz_style_set(style = "research_note", context = "report")
```

Arguments

`style`	Named style family.
`context`	Output context such as ‘"report"‘, ‘"slide"‘, or ‘"dashboard"‘.

Value

Invisibly returns the resolved style object.

Examples

```
viz_style_get("policy_memo", "report")
viz_palette_get("institutional_blue")
local({
 old <- options(investlabr.viz.style = "research_note", investlabr.viz.context = "report")
  on.exit(options(old))
  viz_style_set("research_note", "report")
})
data <- data.frame(date = as.Date("2025-01-01") + 0:9, value = 1:10)
p <- ggplot2::ggplot(data, ggplot2::aes(date, value)) + ggplot2::geom_line()
```

```
p <- viz_theme_apply(p, style = "policy_memo", show_compiler = FALSE)
p <- viz_annotate_last_value(p, data, "date", "value")
p <- viz_annotate_event_lines(p, data.frame(datetime = as.Date("2025-01-05")))
p <- viz_annotate_regime_bands(p, data.frame(
  start = as.Date("2025-01-02"), end = as.Date("2025-01-04")
))
viz_annotate_source_caption(p, "Synthetic data", "Illustration only")
```

`viz_theme_apply` *Apply a standardized investlabr plot theme*

Description

Apply a standardized investlabr plot theme

Usage

```
viz_theme_apply(
  plot,
  style = NULL,
  context = NULL,
  legend_position = NULL,
  base_family = NULL,
  show_compiler = TRUE
)
```

Arguments

`plot`	A ‘ggplot‘ object.
`style`	Named style family or ‘NULL‘ to use the package default.
`context`	Output context or ‘NULL‘ to use the package default.
`legend_position`	Optional legend position override.
`base_family`	Optional font family override.
`show_compiler`	Whether to append the configured plot compiler name to the caption.

Value

A styled ‘ggplot‘ object.

Examples

```
viz_style_get("policy_memo", "report")
viz_palette_get("institutional_blue")
local({
 old <- options(investlabr.viz.style = "research_note", investlabr.viz.context = "report")
  on.exit(options(old))
```

```
  viz_style_set("research_note", "report")
})
data <- data.frame(date = as.Date("2025-01-01") + 0:9, value = 1:10)
p <- ggplot2::ggplot(data, ggplot2::aes(date, value)) + ggplot2::geom_line()
p <- viz_theme_apply(p, style = "policy_memo", show_compiler = FALSE)
p <- viz_annotate_last_value(p, data, "date", "value")
p <- viz_annotate_event_lines(p, data.frame(datetime = as.Date("2025-01-05")))
p <- viz_annotate_regime_bands(p, data.frame(
  start = as.Date("2025-01-02"), end = as.Date("2025-01-04")
))
viz_annotate_source_caption(p, "Synthetic data", "Illustration only")
```

Index

brief_data_as_of, 3
brief_plot_registry_entry, 4
brief_plot_registry_entry_v3, 5
brief_plot_registry_validate, 8
brief_plot_registry_write, 9
brief_plot_registry_write_from_meta, 10
brief_score_label, 12

eval_event_performance, 12
eval_portfolio_performance, 13
eval_strat_performance, 14
eval_strat_plot_scatter_maxdd_annret, 16
eval_strat_plot_tsline_eq, 17

factor_bounded_score, 18

gen_candle_plots_with_sr_dts, 19
gen_candle_plots_with_sr_lines, 20
gen_facet_plot_from_multicol_ts, 22
gen_grid_of_plots_with_labels, 23
gen_plot_comparing_events, 24
gen_plot_event_tsline_cum_ret, 25
gen_rolling_monthly_stats, 26
gen_yield_curve_plot, 27
gen_yield_curve_plot_grid, 28
get_optimal_weights, 29
get_yield_data_DT, 29

prep_backtest_result_from_equity, 30
prep_fill_forward, 31
prep_ratio_ohlc, 32
prep_recent_changes, 33
prep_series_wide, 33
prep_zscore, 34

sim_ar1_dgp, 35
sim_ar2_dgp, 36
sim_ar3_dgp, 37
sim_arch1_dgp, 38
sim_forward_fan, 39
sim_garch11_dgp, 40
sim_ma1_dgp, 41
sim_scenario_path, 41

viz_annotate_event_lines, 42
viz_annotate_last_value, 44
viz_annotate_regime_bands, 45
viz_annotate_source_caption, 46
viz_candle_base, 47
viz_forward_fan, 48
viz_palette_get, 49
viz_series_lines, 50
viz_style_get, 51
viz_style_set, 52
viz_theme_apply, 53